Internal communication in multi-company projects

Mikhail Kapitonov

University of Tampere
School of Information Sciences
Computer Science
M.Sc. thesis
Supervisor: Timo Poranen
July 2014

University of Tampere
School of Information Sciences
Computer Science / Software Development
Mikhail Kapitonov: Internal communication in multi-company projects
M.Sc. thesis, 44 pages, 2 index and appendix pages
July 2014

In global economy software development has to adapt to changing customer demands very quickly. In some cases projects are too big for one company to handle or involve using several fundamentally different technologies. Strategic partnerships between software companies are a widely accepted solution in those situations. When dealing with software development teams spanning across several companies project managers have to solve many co-operation and coordination issues. The co-operation issues in such projects are often closely similar to those described in papers on Global Software Development. The purpose of this thesis is to analyze some of the related studies on Global Software Development and formulate main issues of multi-company software development. Additionally, this thesis analyses papers on company co-operation and customer relations. The scope of this thesis is limited to projects in which participating companies are equal partners. The research is further limited by selecting a few problem areas for analysis: managing resources, reporting, coordinating between teams, managing customer relations and team inter-dependency. Multi-company development issues are gathered by analyzing the available materials on multi-site development, inter-company co-operation and customer relation management. The resulting data is then compared to empirical observations in the case study chapter. Since the research area is limited and the materials on the topic are not covering all possible multi-company development issues, some future research directions are outlined in the discussion chapter.

Key words and terms: computer science, software project management, communication, multi-site projects, multi-company projects, knowledge management, customer relations.

Contents

1. Introduction

1.1. Problem background

Software project management takes central place in most of the projects that require building and maintaining a software system. With rare exceptions, this requires having a centralized management structure with developers and specialists at the bottom and managers at the top.

Software project management revolves around analyzing and formalizing methods of controlling and maintaining the development process. The quality of the management work is directly correlated to the project outcome. While it is impossible to guarantee the perfect implementation of the project goals one can certainly say that without strong and efficient leadership and control the software project is doomed to fail. The projects that benefit from a qualified managerial lead can usually fail because of different reasons most of which have to do with lack of resources whether it would be man-hours or physical resources. However, if the managerial work is faulty the project will most likely fail because of that. In fact, in most cases even spending more than enough will not save a project with bad management.

The Project Manager is usually someone who:

- Meets the stakeholders,
- formalizes the project goals,
- plans the project stages,
- allocates resources,
- monitors the project,
- and steers the project if necessary.

A good project manager is always concerned about the impression he and his team make on the customer. In the end, the software project, much like any other project, is successful only if the customer says it is. Customers provide the initial kick for the project and essentially supply it with resources. It is important both to hold a convincing kick-start meeting and to host successful demos and presentations as the project moves ahead. In a lot of cases customers are genuinely interested in project's progress and are eager to take part in technical demos and even in the technology discussions with developers. While it is less common for the customer representatives to contact the team directly, the leading manager should make a note that the possibility to see the people behind the project implementation may greatly increase the trust between the customer and the software company.

In the scientific papers customers are usually presented as stakeholders, but the term ‘stakeholders’ does not only include paying customers, it has a much broader meaning. A stakeholder is a person, an organization or other entity, which is interested in the project outcome or a byproduct of the project outcome. In this definition, the customer, the management and the team are all benefitting from the project in several ways. The project outcome is of direct interest for customer, top management and the team manager. The positive outcome lets the customers achieve their business goals and establishes a positive reputation for the management level and for the software company as a whole. The negative outcome is similarly reflected on both customers and the software house.

The team, which develops the software project, is in a lot of cases a very much involved stakeholder. They are almost as much interested in the project itself as in the salary that they are getting. The project topic must be something the team is interested in and the assets or tools generated while developing the project might be reused in the future. In some companies software teams get a special day in which they are free to develop whatever they like and sometimes those hobby projects end up being sold as real products. One of the examples of this approach is Google [Baldwin, 2012] in which several hobby projects have become noticed and adopted as main products of the company, though this policy might be seeing a decline in recent years. Most important benefits for the team are the experience gained and the tools or methods developed. A project failure can decrease team spirit and make some people consider leaving the company.

To achieve its goals software projects, like other projects, have to be carefully planned out first. One of the unique features of the software project management is that at the planning stage a number of quite different implementation models could be chosen. In fact the planning stage itself could be organized and executed in a number of different ways. Initially, software development followed the implementation model chosen for most of the other industries. This model is now known as waterfall model and is quite straightforward and easy to understand for a technical engineer. The heart of this classic model is an attempt to divide the project implementation in several big stages. The development work in most industries can be roughly split into three categories: planning, manufacturing and exploitation. The actual stages the waterfall model defines are: requirements specification, architecture planning (design), implementation (coding), integration (deployment and taking into use) and maintenance. As the software development industry evolved and became less dependent on the actual hardware vendors it became apparent that one development model would not fit all of the cases. In fact, the virtual nature of the software product made it easier to prototype and test different revisions of the code as long as all stakeholders were

willing to spend resources on that. It soon became apparent that if the prototyping starts early in the project development and the customer is deeply involved in testing those prototypes, the resulting product will likely be much closer to the customer requirements.

The success of the software project is in a lot of cases determined by how well the resources are managed. A good project leader needs to know exactly how many people he will need and what kind of expertise is required from the team members. It is also extremely important to cover for any unexpected events that may lead to delays in the project plan.

Managing resources other than man-hours is usually a simple process. It still needs to be planned and its own paper trail, but it is very likely that physical resource allocation will be determined at the beginning of the project and will likely not change until the very end of it. In a lot of cases software projects will require a minimum of actual physical resources to be allocated and even those will mostly be the form of licenses for specific software products used in development process.

Human resource management on the other hand, is in a lot of cases a chaotic process, especially with smaller and newer companies. Managers may try to keep the availability of human resources in their heads rarely using anything more complex than sticky notes or shared drawing boards. At some point, the system becomes so complex that management has to make a company-wide decision to use a certain method to keep track of it. Most companies end up with some kind of software solution to keep their human resources in check.

It is especially hard to manage human resources in large projects or in projects developed in several physical locations. In cases when the development team is split across several offices, the project manager has to consider communicational overhead when estimating project completion times.

Assembling a capable team and reserving right amount of man-hours is crucial, but then comes the problem of overseeing the progress of the project. Monitoring projects regularly and checking on fulfillment of individual tasks is extremely important, especially for medium-sized and big companies. When the management structure has several levels, the role of the project manager becomes more clear and separated from the roles of team lead or scrum master. Project manager usually will have several projects that need attention, and will not be able to be deeply involved in any of them. The role of the project manager here is to organize the work process rather than to keep track of individual tasks. The bigger the company gets, the more distinct are the roles of the project manager and team lead in the company.

With this in mind, how does the busy project manager keep track of every project he is in charge of? The answer is: through reporting and issue tracking. There are multiple software solutions that help manage projects, time spend, arranged meetings and documentation written. Those systems are well-designed and are useful not only for keeping the manager's life organized, but also for providing required information to his superiors.

Reporting systems can be designed with different project size in mind. Some systems like Atlassian Jira [Atlassian, 2014] are able to track project's progress from individual issues to user stories and product backlogs. Other systems deal more with how busy a person in a team is. Issue tracking systems can also be used to share information with customers and provide them with means to track project status and comment on the progress. In some cases customers are even given an option to create issues in the tracking system and upload appropriate materials to help the development process.

An important part of manager's work is keeping his superiors informed about the current state of the project and its progress. This is usually done via email or some kind of document management system, such as Confluence [Atlassian, 2014] or Alfresco [2014]. Those reports usually contain an overview of time spent on the project and remaining user stories to complete. The document processing and collaboration systems are used not only by managers but also by developers to keep project-related documentation ordered and indexed for search. In some cases development team members are also required to analyze project status or estimate future work for the project manager, since they have better grip on the possible timeframes of their work.

Even with best tools and resources available, there could be some situations in which the project is in danger of being delayed or even cancelled. In those cases, project manager arranges special meetings in which team tries to figure out measures to overcome the crisis. Usually such measures involve re-prioritizing tasks, requesting additional workforce, dropping minor features or in worst-case scenarios abandoning the project. Those special meetings are called steering meetings and are a defense mechanism against immediate project failure.

In modern world many software companies have long outgrown startups they once were. In many cases handling the development of large projects such as operating systems or media suites requires having a big staff and careful release planning. Large teams are in a lot of cases spread across the physical offices and multiple countries.

Even if the company handling the project development is a small one, it still has to consider the globalization trends in their work. In many cases, the customer side is represented by a much larger business entity than the software house and the project manager has to understand how to properly present project progress and current issues to the customer representatives in accordance with their position in their company.

There are cases in which the project manager will have to contact representatives of higher management, members of the ICT department and the management of end-user teams on the customer side. All of those stakeholder groups have different agendas and may have slightly different views on the project they are ordering.

Large companies that are sharing project workload across several offices have to deal with a lot of communication and synchronization issues. These problems are recognized and have been investigated by both scientific society and privately by in-house coordination specialists. A similar set of problems often can be found in cases where a number of small companies are undertaking one project while sharing the workload. In this case you also have to deal with communication and synchronization problems, but you have to take into account that you do not have a unified command structure. When synchronization difficulties arise in multi-site development, one of the most common ways to overcome them is to escalate the issue to the higher level of management, common to all involved teams. In case where you do not have a common management structure, you no longer have that opportunity. There are a number of other important differences between multi-site and multi-company development that will be discussed in this thesis.

1.2. Research questions

The purpose of this thesis is to analyze the problems of multi-company software development and project management. The main areas of the analysis will be the communication between the companies and project teams and their relationships with customer. By performing this analysis and formulating the main issues project managers encounter in multi-company projects we can also outline possible ways to overcome the biggest obstacles in such projects.

This thesis will touch on the following areas of managing a multi-company project:

- managing resources,
- reporting,
- coordinating between teams,
- managing customer relations
- and team inter-dependency.

This thesis tries to answer the following research questions:

- What are the main issues of multi-company software development in selected areas if participating parties have equal standing?
- Are the solutions mentioned in the papers on the multi-site development applicable to the multi-company development?

1.3. Research methods

The main research method chosen for this thesis is the literature review of materials available on multi-site software project management and inter-company relations. The first research question will be answered through the literature review. The second research question will be answered using a combination of a literature review and some empirical observations in the case study chapter.

1.4. Thesis structure

This thesis is structured in a way that enables the introduction of the subject matter and helps provide relevant connections to the previous research on the subject. First half of the thesis introduces the concept of multi-site software development and its role in the globalized world. Second half of the thesis extrapolates on those issues using relevant research in business relationships and Hofstede's cultural dimensions.

The first two chapters of the thesis will introduce the research area and provide sufficient background information. Multi-site project development will be described in Chapter 3. Chapter 4 will list known problems in multi-company projects, the purpose of this chapter is to use information gathered in Chapter 3 and expand on it using the available research on multi-site projects, outsourcing and business relations. Chapter 5 will contain the example cooperation scenarios in multi-company projects. The last chapter will use the information gathered in Chapters 4 and 5 to briefly outline problems of multi-company project development and their solutions.

2. Software project lifecycle

To begin tackling the project management issues we have to first answer the question: what is project management? The Open University definitions of management are state that management is a collection of the following activities [Hughes and Cotterell, 1999]: planning, organizing, staffing, directing, monitoring, controlling and representing. Those activities are usually performed by a group of specialists, who know exactly what to do when the project they are overseeing encounters difficulties. In the modern world we would usually see larger companies performing bigger and more sophisticated projects. In those companies there is usually an entire system of management in place that takes care of various areas of project life cycle. The managerial system will have multiple layers, but usually there is one person responsible for the success of each project.

Problems that managers have to solve arise mostly from communication difficulties. While project managers can name lack of proper standards, tools or estimates as the main issues they face in their work, team members define the lack of proper communication as the main reason a project could fail. Almost all of the issues mentioned by managers, project team members and customers [Hughes and Cotterell, 1999] are likely a result of miscommunication of those parties at different stages of the project. As an example one can say that changing requirements or deadline pressure could be likely caused by the fact that customer has not properly communicated all requirements at the beginning of the development cycle.

There are many different approaches to software project management, for example:

- Standard or waterfall model,
- Spiral model,
- Extreme Programming,
- Rapid Application Development,
- PRINCE2,
- Feature Driven Development, and
- Scrum.

This list is of course incomplete as there are not only many other methodologies but also combinations and adaptations of those. Agile methodologies are becoming increasingly popular in recent years with Scrum leading the pack.

Waterfall model [Benington, 1983] was mentioned in the introduction chapter already as it is the most natural way to develop something, whether it be a software solution or any other type of product. As was already stated, waterfall model is derived from the basic lifecycle of any industrial product and is used in many other tech industries. Spiral model extends the waterfall model by introducing prototyping and versions. It has in most cases replaced waterfall model especially for large projects.

Extreme Programming [Computerworld, 2001] is a methodology, which aims at lowering the cost of changing the requirements. In Extreme Programming releases are frequent and development cycles are short. Features are only implemented when needed and no design specification documentation is generated. Extreme Programming introduces development methods like frequent code reviews and pair programming.

PRINCE2 (acronym for Projects In Controlled Environment version 2) is a process-based [PRINCE2, 2014] project management methodology. It is based on eight high-level processes, which define actions to be performed in each stage of the development process. PRINCE2 differs from most agile methods by concentrating more on setting strict ordering rules that the project development should follow than trying to adapt to changing requirements. PRINCE2 is suitable for managing projects in clearly defined framework and can be considered inappropriate for small projects or projects with changing environments.

Rapid Application Development [Martin, 1991] is an approach focused on fast and early prototyping. It may use only minimal overall planning and allow software system to be written much faster. Special RAD software development tools available on the market are centered around ready-made components used as building blocks for the application. The availability of those pre-made components made RAD development environments especially popular for implementing small helper applications with standard UIs.

Scrum [Scrum, 2014] is a strictly defined project management approach, which adds several new terms and roles to the development process. In a lot of cases teams will not use all of the Scrum tools and methods, but rather take what they really need or find useful.

Feature Driven Development [Palmer and Felsing, 2002] is as its name implies, a methodology focusing on delivering features. It uses small iterative steps to deliver end-user features, which should take no more than 2 weeks each to complete.

Most of the software houses are familiar with at least several of the software development methodologies and will adapt those to their practical needs. Since each of the methodologies have their own strengths and weaknesses the selection of the one used to implement a project is a process that requires careful consideration. Most of the methodologies were actually developed by big companies performing a unique project and were tailored for that project.

Another option that software houses tend to pick is the partial adoption of some methodology. In this case only certain elements or procedures described in its guidelines will be used, while trying to retain the overall spirit of the methodology.

While determining the appropriate methodology from project implementation is important and wrong decision can lead to project failure, software projects have other problems as well.

Thayer et al. [1981] outlined the following project management problems in their report:

- poor estimates and plans,
- lack of quality standards and measures,
- lack of techniques to make progress visible,
- poor role definition, and
- incorrect success criteria.

Let us shortly elaborate on these issues. The most obvious and commonly mentioned software development issue is the lack of estimation and proper planning. In fact, this issue is named as the top culprit of project failures by various studies. In fact, it is nearly impossible to complete a project without a proper completion plan and that requires well-defined estimations on how long will it take to complete it. Effort estimation is an extremely difficult problem, which many software development methodologies solve in its own unique way. Scrum for example [Scrum Institute, 2014] uses a technique called “planning poker” or “scrum poker” to estimate the effort of implementing a certain feature. It is based around a deck of cards with different completion time estimations. Each team member selects the estimation on their own and then all of them show the cards. If the estimations don’t match, team members need to explain the reasons for selecting their estimate and another vote is done. In other cases, effort estimation might be solely based on individual developer’s experience with implementing similar features. This usually happens if the team is fragmented by areas of expertise and you only have one developer per technology area (for example: one UI developer, one backend developer and one scripting specialist).

Lack of quality standards and measures can turn even well planned projects in a disaster by making the final product unstable and as a result – unusable. In software projects there are many things for which quality control is required. While most developers will be primarily concerned with code quality and try to tackle the problem with testing, the quality issue goes well beyond that. Aside from validating code quality and standards, for which companies are usually automatic utilizing semi-automatic check systems, there could be need to enforce documentation standards, issue tracking

standards and overall work quality standards. Team lead would usually be more involved in enforcing the coding standards and project manager would mostly try to enforce documentation and reporting standards.

Phan et al. [1995] defined several key factors in improving software quality in distributed projects.

1. Well-defined quality goals and objectives that are driven by market and end-user needs.
2. Good management of reusable and scaffolding code.
3. Good quality assurance planning and control.
4. Effective feedback.

In the wider analysis by Nguyen-Duc et al. [2014] it was revealed that global dispersions (which are a type of distance between teams) have varied implications on team performance and software quality. In particular, while geographical dispersion has a bad impact on software quality in general, temporal dispersion has that impact mostly in closed-source projects.

The visibility of the project progress is also an important issue. In a lot of cases customer would want to see the progress that the team has made since last development iteration. Some software projects have difficulties in this regard. Most commonly there is a situation when the visual changes are only recognizable at the latest stages of development. There could be also situations in which there are no intended visual changes and you need to explain the internal changes to the customer to get their approval. Verner et al. [2014] mention the risk of losing track of the project progress as one of the control risks found in distributed software projects. The mitigation advice that is outlined from their analysis is to define clear roles and responsibility areas at the beginning of the project.

Poor role definition can cause various issues in the course of the project ranging from power struggle to inability to assign tasks. Pikkarainen et al. [2008] state that the lack of proper role definition may cause significant problems in producer-consumer relationships within the project. Project manager needs to specify who is responsible for which areas and tasks in the project and help maintain order and proper work distribution.

Incorrect success criteria is the usual predicament in projects where both the customer and the project manager lack experience in properly defining project goals. This problem usually becomes evident close to the end of the project when project managers and customers cannot agree on the level of completion of the planned features.

Understanding those issues and the general procedure of developing a software product provides a framework for further formalization of the software development practices and helps improve their efficiency. There are, of course, many other research areas that one must consider when analyzing software development practices and project management techniques. In this thesis only a subset of those topics will be covered with the main attention to human communication and human resource management.

Managing human resources in software project is one of the key issues of software project management. The selection of right people with right expertise is crucial to the success of the software project and can either delay the project or significantly speed up its completion.

3. Multi-site projects

As the information technology industry matures, there is a growing need for complex software systems. The advancement of electronic document processing, e-government, data mining and many other breakthrough projects requires software development houses to innovate faster and involve larger teams. In a lot of cases this means that the companies will have to invest in Global Software Development. Global Software Development or Distributed Software Development is defined as a process of creating software in which team members are distributed around the world [Prikladnicki and Audy, 2010].

Globalization of the software industry, as with many other industries, is driven by economics. It simply becomes more efficient to create software with global markets in mind rather than trying to make an adapted solution for each country from scratch. The effectiveness of creating international software versus multiple localized versions is only one of many reasons why companies are considering GSD.

More than a decade ago many companies have realized that to decrease cost of skilled labor they have to experiment with remote labor markets, which lead to creation of the first outsourcing offices in software industry.

Outsourcing, as outlined by Herbsleb and Moitra [2001] enabled companies to overcome difficult business challenges such as:

- lack of skilled human resources,
- need to be in close proximity to the target markets,
- ability to quickly form virtual teams to exploit market opportunities,
- short time-to-market by enabling 24 hours a day development in different time zones, and
- the ability to merge and acquire companies as soon as there is an opportunity.

Global Software Development is also quite challenging. Hebsleb and Moitra [2001] outlined several problems that exist in distributed projects:

- strategic issues,
- cultural issues,
- inadequate communication issues,
- knowledge management issues,
- project and process management issues
- and technical issues.

Strategic issues stem from the need to divide tasks of certain type to specific locations or company branches. Deciding how to do that is quite difficult as companies are usually constrained by the resources available at those sites, their levels of expertise, infrastructure, proximity to customers and other things. Another organizational issue mentioned in the article is the fact that big companies tend to resist changes, which is usually caused by either miscommunication between management levels or the fact that specialists believe that those changes might cost them their job.

Cultural issues play an important role in multi-site software development, especially in multinational companies. The difference between work cultures, attitudes, hierarchical behaviors can be hard to adapt to and will likely cause worker anxiety or more serious problems.

Hofstede [2010] outlined several cultural dimensions, which can be used to explain differences in behavior in multi-site projects:

- power distance,
- collectivism vs. individualism,
- femininity vs. masculinity,
- uncertainty avoidance,
- long-term orientation
- and indulgence vs. restraint.

Power distance is defined as the extent to which different junior members of power hierarchy accept that power is distributed unequally. In high Power Distance cultures inequalities are expected and tolerated, teachers take initiative in class and quality of learning is dependent on the excellence of the teacher. In low Power Distance cultures inequalities are considered unjust and are minimized, teachers expect initiative from the students in class and quality of learning is a result of two-way communication and excellence of students. In multi-site development low power distance cultures are harder to manage, they require much better communication levels to keep the responsibilities synchronized. In high power distance cultures employees tend to know

their role and place in the project really well, which makes it much easier to organize such teams over long distances.

In *Individualistic* societies the ties between individual members are loose, everyone is expected to look after themselves and individual achievements are valued more than group achievements. In *Collectivistic* societies the decisions are usually made in the organized manner and an individual is expected to work towards a common goal. For multi-site development this dimension determines how easy it is for the team to retain the team spirit when working separately. Teams in Collectivistic societies will likely have some level of common understanding since the beginning of the project and will maintain closer ties with their colleagues later in the project lifecycle.

Feminine societies value relationships and quality of life, while *Masculine* societies encourage competitiveness, assertiveness, materialism, ambition and power. In Masculine cultures the difference between gender roles is more pronounced, while in Feminine cultures men and women share the same values with the stress on modesty and caring. The relation feminine-masculine dimension has a more subtle influence on the multi-site project development. The influence it will cause will depend on the actual team composition and other stakeholders.

In high *Uncertainty Avoidance* cultures people tend to show more emotions, they believe that "what is different is dangerous". In low Uncertainty Avoidance cultures people are calmer and adapt well to the unexpected events, regarding them as normal part of life. Since multi-site development requires at least some degree of standardization and planning, employees in high Uncertainty Avoidance cultures will feel more at ease. However, in complex agile project it would be more beneficial to have personnel with higher affinity for solving unexpected issues.

Long-term orientation is a dimension, which describes how the society deals with planning and distant goals. Long-term oriented societies emphasize the future, they have pragmatic values and are oriented towards persistence, saving and capacity for adaptation. Short-term oriented societies value their past and present. They are interested in keeping the traditions, preserving one's face and fulfilling social obligations. In multi-site software development this dimension will not have that much of an impact, as the need for overall planning will balance out the culture differences.

Indulgence vs. Restraint is a dimension, which defines a level to which members of a society try to control their desires and impulses. Indulgent societies have higher tolerance towards basic human desires such as having fun, relaxing and enjoying the life. People in restrained societies believe that basic desires should be restrained and regulated by strict norms. In multi-site development this dimension does not have a significant impact if the sites share the same cultural background.

Hofstede [2010] also defines different organizational-level culture dimensions that were derived from IRIC cross-organizational study:

- process oriented versus results oriented,
- employee oriented versus job oriented,
- parochial vs. professional,
- open system versus closed system,
- loose versus tight control
- and normative versus pragmatic.

Process oriented organizations tend to concentrate on procedure of implementing something while the result-oriented organizations concern themselves more with actual goals. In *result-oriented* organizations people tend to be more confortable towards encountering unfamiliar situations, while process-oriented organizations tend to value standardized way of doing things. Many software companies have result-oriented cultures, but when the companies grow in size or start doing multi-site development there is a growing need for formal procedures. This need forces many companies to shift their culture to a more balanced approach. The actual balance between process-oriented approach and result-oriented approach is also dependent on which type of projects an organization has in its portfolio. Hofstede [2010] stated that one of the organizations observed with highest process orientation was drug manufacturing. This can be explained by the fact that in pharmaceutical industry there has to be clear planning and extremely thorough quality control on all steps of drug manufacturing process.

Employee-oriented companies are more concentrated on keeping the personal wellbeing of their workforce in check than achieving better results in production. In *job-oriented* companies people often experienced a large strain and they often felt that actual work is more important for the organization than their personal and family life. IRIC study shows that on the management level there are different opinions about whether the organization should be more employee-oriented or not. It is mostly related to the fact that job-oriented culture may be more efficient and productive than employee-oriented. In corporate world, it is usually up to the top management to choose the human resources strategy and this strategy will be enforced on all of the company branches. In multi-site development the impact of this dimension is questionable and may be dependent on actual project types.

Parochial vs. professional dimension describes how organizations influence the personal identities of their employees. Members of parochial cultures believe that organization norm cover both their behavior at work and at home. They assume that when their company hires an employee they take the employee's social and family background into account as much as their work competence. Members of the

professional cultures however, believe that their personal life is their own business and that they should be treated at work according to their competence level. In multi-site development this dimension does not influence the project success rate by itself, but may lead to different results.

Open system vs. Closed system dimension compares companies with a welcoming environment towards newcomers to those, which are more hostile or secretive towards them. In multi-site development open companies will likely have an advantage, as introducing new people from various remote locations will be much easier.

In companies with *loose control* culture developers tend to have more decision-making power. This is especially true for software houses specializing on agile development methods. Companies with *tight control* cultures have an easier time dealing with big projects and corporate customers because they can relate more on the managerial level. In multi-site development tight control cultures will have an upper hand because it is much harder to manage decision making in a multi-site team.

Companies with *normative* cultures are prioritizing their view on the market and the company standards, while companies with *pragmatic* cultures are willing to bend the rules to a certain extent to ensure better profit and good customer relations. With the advent of customer relationships management (CRM) there is a growing trend towards having a customer-oriented business model and in software industry most companies have adopted the customer-oriented approach. In multi-site development this dimension does not have much influence on the project management, other than that it already has for regular projects.

In terms of communication multi-site projects share the same difficulties as the regular software development projects. The communication gap and the sheer size of those projects make regular software project management issues harder to deal with.

Knowledge management issues in multi-site projects stem from the fact that while in co-located teams you can have peer-to-peer knowledge transfer, in multi-site teams that is often impossible. Knowledge transfer in multi-site environment requires specialized software solutions and some degree of standardization.

Project and process management issues are causes by the fact that teams in different locations can view certain procedures differently. Synchronization is extremely important and is commonly defined via milestones and clear entry and exit criteria.

Technical issues are various technology-related problems that are hard to manage across several locations. In multi site teams you have to carefully plan and execute tasks such as transfer and storage of confidential information, configuration distribution, file sharing and other similar activities.

Let us also try to analyze how the multi-site project environment affects software project management problems, mentioned in the previous chapter. The work effort estimation in a multi-site environment is a more challenging task than in the regular team. The project manager needs to account for travel and distance issues when dealing with a distributed team. There will be more travel and video conference calls in a multi-site team and the time spent for that needs to be accounted for in the estimations.

Managing standards in a multi-site team will require using special tools and software. The standards have to be documented and stored in the known location that is accessible by all team members. There are document storage and collaboration facilities designed around the idea of a multi-site environment. Atlassian Confluence [Atlassian, 2014] and Alfresco [2014] are two examples of such tools. Knowledge management issues are increasingly important in larger companies and especially in global companies. In software companies there is always a need to have an information sharing system to store both development-related documentation and information from customers. The effectiveness and accessibility of such system becomes extremely important in situations where those documents cannot be otherwise reached.

Progress visibility in a multi-site environment can also be a bigger issue than in the regular team. Since some of the team members do not have a way to casually contact the others, some of them can start doubting the importance of their colleagues. To make progress more visible and to underline the importance of all participating team members project managers should carefully review each completed task together with the team.

Role definition issues are tackled the same way as they are in the single-site teams. In multi-site teams, it is much easier to loose track of the project roles and responsibilities, so there might be a need of a special document that outlines the roles and is accessible by all team members.

Work distribution and coordination issues are the major threat to the multi-site project's success. The article by Mockus and Weiss [2001] suggests that work is to be split in the individual units they call "work chunks". Mockus and Weiss state that software developer's work can be split into work items that range in size from a small patch to very large set of changes. There's also a difference in the importance of the work item and in how it is going to be used later.

They have defined the following set of work items:

- releases,
- features,
- modification requests,
- and deltas.

Releases, that can also be called "customer deliveries", are a set of features agreed with the customer and scheduled for completion within a certain timeframe. *Feature* is an individual piece of functionality, usually associated with the customer's user story.

Features consist of one or several *modification requests*, which are in turn a set of *deltas*. Mockus and Weiss argue that modification requests are a unit that can be used to measure and manage multi-site development efficiently. Deltas are the minimal work unit in this system, an individual code commit.

One of the main tasks of multi-site development is ensuring a proper distribution of work across several sites. Mockus and Weiss propose that when the project manager decides on transferring a development of a feature to another site, the following approaches can be considered:

- transfer by functionality,
- transfer by localization,
- transfer by development stage
- and transfer by the maintenance stage.

Transfer by functionality is a process in which the ownership of a subsystem is transferred. Distributing the knowledge and development by functional area is going to specialize each site and task it with taking care of a specific part of the developed system. The good part of this is that any change in a specific problem area will only require a small team of experts. On the other hand, implementing a new feature that requires knowledge from several such areas is going to be a harder task because it might require experts from several sites to coordinate their effort.

Transfer by localization is a process in which a subsystem or a feature is tailored for the local market by a team with the closest proximity to it. Localization or country-specific UI changes are good examples of such tasks. The disadvantage is that if the modifications require something more than cosmetic changes you will need to maintain knowledge area experts on site, which is quite inefficient.

Transfer by development stage is an approach in which developers perform different development stage activities in different locations. For example the UI design can be done in one office, planning in another and testing in a third one. The advantage in this approach is having well-trained teams that are efficient at certain development stage activities, such as maintaining the system or designing the architecture. The

disadvantage is that some coordination effort will be needed to ensure that project progresses through the development stages smoothly.

Transfer by maintenance stage is a method in which development of the older releases, which are primarily in the maintenance phase, is transferred to a specialized location. This makes more resources available for developing new releases, but the disadvantage here is decrease in quality and increase in problem resolution intervals. This happens because the site supporting the old release did not take part in the development of it.

Mockus and Weiss suggest that the actual work distribution needs to be decided based on the following factors:

- work coupling,
- amount of effort
- and learning curves.

When considering work coupling factor, project manager needs to determine which work items are heavily inter-dependent and try to make sure those items are assigned to one location. Having as few co-dependent work items as possible is extremely important. In addition to assigning the co-dependent tasks to the proper sites efficiently, project manage needs to consider existing coordination effort overhead and make sure to not put too much coordination strain on each site.

Different work items can require different amount of work effort from each site. The project manager needs to carefully analyze the work effort required by each site to implement the work item and select the appropriate site based on that. The required effort can be estimated using historical effort trends.

When a work task is transferred to a site, which lacks the expertise to undertake it, it will likely cause significant delays in the implementation. Developers will have to adjust their effort estimations appropriately when facing area of expertise they are not familiar with.

Based on previously mentioned studies we can re-affirm the fact that with the introduction of several locations, software development is burdened with more co-operation and synchronization issues. However, most of these issues have solutions and could be resolved by careful analysis and planning.

4. Inter-company project coordination

Inter-company projects are in the most cases quite similar to the multi-site projects developed by one company in terms of issues that the management has to solve. One major difference that sets them apart is that the management structures of inter-company projects are independent.

Independent management structures usually means that the workload and task division needs to be negotiated beforehand by project management teams from all participating companies and any changes to that initial agreement will require another meeting to be held.

In many cases effort estimation and work synchronization discussion will require the presence of the customer as the working hours are ordered separately from each company and they cannot directly propose anything to the other implementing party.

Hofstede's [2010] cultural dimensions play a more significant role in the multi-company project environments. In the next sections of this chapter Hofstede's dimensions will be used to outline possible cross-company communication issues. While in multi-site development cultural issues could be mitigated by managerial involvement, in cross-company projects this may be impossible since the conflicting parties may belong to different organizations.

Let us also review on how software project management issues are affected by a multi-company setting. The work effort estimation becomes exceedingly troublesome in the multi-site setting, since several business entities have to work together to produce a single estimation. Usually, several meetings are organized between the customer and participating parties to plan the overall implementation. After the overall plan is settled, the individual business entities can hold their own meetings to plan their part of the work. It is extremely important to have a clear architecture drafted out on the first common meeting with the customer. When the overall architecture of the project is defined and it is clear which parts of the project are going to be implemented by which company, it will become easier to manage dependencies between teams. Work schedule is to be drafted after the possible co-dependencies have been accounted for.

Since in multi-company project it might be hard to employ the regular estimation techniques, the initial meetings should have at least some knowledge area experts (senior developers or architects) present to help with the draft estimates. Their role is to account for possible difficulties and delays in the implementation using their expert knowledge. They have to produce a draft estimate that can be later refined on the team meetings in their companies.

Standards and quality are also hard to manage in multi-company environments. How the quality issues are resolved is heavily dependent on how well-integrated the business entities are in their development effort. Having a known, shared place for storing internal documentation is crucial in the multi-company project setting. Usually, such things are agreed upon on the first common meeting, but some of those systems can be put in place later. Since it is not efficient to have a separate document sharing solution for each cross-company project the business entity is participating in, it is often agreed that one of the software houses will host the majority of documentation on their internal system.

The visibility of the progress is yet another difficult issue for the multi-company projects. In multi-site development it is important to make progress visible to the customers and to the team members, but in multi-company environment it is also extremely important that your partner companies see the development progress properly and unbiased. Clean and easy-to-understand progress reports will improve the company relationships with both the customer and with other participating companies. Having frequent progress meetings and reports also ensures that the company is secure from misconduct or inefficiency accusations by competitors.

Role definitions need to be clearly outlined and documented. In some cases developers will have roles that give them some authority across several companies. In this case, the role definition needs to be formalized so that each team member knows about it.

Success criteria need to be defined on the initial meetings and properly documented. Ensuring the proper wording is extremely important to eliminate conflicts and make sure the project will progress smoothly. Changing the success criteria is a big effort and in case of a multi-company project it will require having a joint meeting between the customer and all participating companies. This means that it is crucial to define the success criteria in such a way that discussing them late will not be necessary.

Another important issue in the multi-company project is work distribution. Looking back at the article by Mockus and Weiss [Mockus and Weiss, 2001] it seems that in a multi-company setting, task transferal can really gravitate towards the following options:

- transfer by functionality and area of expertise,
- transfer by location
- and transfer by maintenance stage.

In a multi-company project, the following things can motivate the presence of individual business entities:

- expertise in a certain area of knowledge,
- proximity to the customer,
- and ability to host and support a product.

Aside from those three reasons, there is a possibility that companies are invited in the project because it is too big for one of them individually, but in this case customer will likely go with a bigger software house to alleviate the overhead of supervising a multi-company project.

Considering the above, work items are usually transferred to the company that has bigger amount of expertise on them, or is efficient in completing them by some other reason. For example, in a multi-company project one company can have a lot of UI experts and designers and the other one can have a lot of server and database experts. In this case the first company will be concentrating on the frontend and UI, while the other will be implementing the backend. In another example, if a company A has great development expertise, but no dedicated hosting capabilities or much of the support staff, then the support task is going to be given to company B that has the resources necessary for it. And finally, if the company A has all the programming experts and the company B is closer to the customer and has some UI and localization experts, it is really likely that the company B will receive most of the support and localization tasks.

4.1. Reporting and Knowledge Management

The inception point of any software project is usually marked with generation of a set of documents specifying the content of the planned activities, initial work estimations and implementation schedule. In multi-company projects, this process is more complicated as several documents need to be created which will not only detail the content performed by each participating software house but also provide communication and work synchronization framework for the project. Additionally, those documents are generated by different organizations and on different organizational levels. Some of those documents will never be shared or reused, like individual opinion or reporting emails send by developer to superiors. Others are likely to be processed and pushed to higher managerial level and maybe even shared with other contracting parties.

Reporting issues are quite dangerous and might cause project delays, wrong work estimations and overall miscommunication and mistrust. Most of those issues are caused by improperly or incompletely shared information, but there are also cases in which information is wrongly prioritised causing important data to go unnoticed.

Cultural dimensions take an important role in reporting and knowledge management. On a personal level, different cultures can have different standards on what is important to share with their colleagues and on how to share that important information. On organizational level cultural differences may lead to misunderstandings in interactions between managers and employees from different companies.

On the personal level, different *power distance* level in collaborating companies may lead to communication difficulties when making information requests across organizations. Since different companies may have different understanding on who can contact whom, this will cause irritation or conflicts in worst-case scenario. The *collectivistic* culture can also cause negative effect on multi-company knowledge sharing. While in multi-site development a distributed team with collectivistic culture (given that company is culturally monolithic) will get benefits from closer personal ties, in cross-company projects this will lead to "we" – "they" scenario and make it harder to build trust between team member from different companies. The level of *masculinity* in the culture should not influence information sharing too much, however some issues may occur during peer-to-peer information sharing. Difference in *uncertainty avoidance* level might cause some issues since companies will likely have different views on the level of detail and standards of shared documentation. *Indulgence vs. Restraint* cultural dimension differences will only cause issues in peer-to-peer information transfer.

On organizational level only certain cultural dimensions will have a noticeable influence. Some of the dimensions such as *employee vs. job orientation* are not so visible to the employees of collaborating company. Most important cultural dimensions here are *open vs. closed system* dimension and *normative vs. pragmatic* dimension. The more *open* the organization is the more actively it will share relevant knowledge to co-operating parties. Software companies tend to gravitate towards pragmatic, customer-oriented cultures. Problems may occur in cases where a company with pragmatic culture has to co-operate with another entity, which has normative culture. This can lead to various tensions due to differences in operation methods between the two entities. This will complicate further if both companies have a normative culture since the process of negotiating the communication norms will be more difficult.

In multi-company environments each company usually organizes knowledge separately. This may create difficult situations, such as having a set of documents in multiple copies in different organizations. This increases the workload of the programmer teams since they have to synchronize their reports across two or more issue management tools.

Dingsøyr and Šmite [2014] define the following approaches for knowledge storage and gathering:

- systems school,
- cartographic school,
- engineering school,
- organizational school,
- and special school.

Systems school focuses on gathering and storing knowledge in repositories. In a repository the knowledge is standardised and systematic, which works well across geographic and temporal distances. However, there can be sociocultural challenges such as different preferences on the level of detail and importance of knowledge.

The cartographic school is centred around knowledge maps and knowledge directories. This approach enables learning both at individual and company level. It is most effective when the temporal distance is small and the knowledge could be transferred orally. This school can also be vulnerable to sociocultural distance as there can be variations in what kind of knowledge is mapped and how it is represented.

Engineering school is focused on processes and knowledge flows within organizations. This approach is described as focusing primarily on processes for mapping knowledge, conducting project retrospectives, mentoring programs, and describing work processes. Since this approach is based on explicit knowledge, it is not affected by temporal or geographic distance. There are still possibilities, however, of knowledge about processes being interpreted differently in various sociocultural environments.

Organizational school focuses on organizing expert networks for knowledge sharing. In such networks both tacit and explicit knowledge can be shared. Since knowledge is usually transferred orally, this approach can suffer from challenges related to geographic, temporal, and sociocultural distances.

The spatial school is focused around organizing knowledge in a physical location. This can take form of setting up publicly available whiteboards for sharing important information or taskboards for sharing project status and task assignment. This knowledge management strategy is highly affected by geographical and sociocultural distances.

Dingsøyr and Šmite [2014] note that traditional global projects mostly rely on systems and engineering schools, because formalized explicit knowledge is much easier to do over distance. They also state that since agile gravitate heavily towards sharing tacit knowledge and that implies dominance of special and organizational schools. Cartographic school is considered to be a cost-effective solution for both traditional and agile global projects.

In the paper "Managing Knowledge in Global Software Development Efforts: Issues and Practices" [Desouza et al., 2006], authors define the following strategies for dealing with knowledge management in global software development:

- headquarters commissioned and executed,
- headquarters commissioned and locally executed,
- regionally commissioned and locally executed.

In the first case, the head office standardises the knowledge management system, provides support and manages the implementation.

In the second case, the head office defines the guidelines for knowledge management, but the actual details and implementation is left to the other offices.

In the third case, each office manages its knowledge separately. This could be beneficial if each office want to tailor the knowledge management approach for local needs rather than take global company structure into consideration.

The paper also lists three knowledge management architecture models, which companies can use with the appropriate strategy:

- client-server model,
- peer-to-peer model
- and hybrid model.

Clients-server model implies a single unified repository for knowledge sharing. Peer-to-peer model implies that area experts will share their knowledge directly with other developers. Hybrid model is a combination of both approaches which both has a repository for knowledge sharing and encourages peer-to-peer information exchange.

When selecting the knowledge management strategy and model, the company needs to know what is the end goal of its knowledge management efforts. If the goal is to ensure easy reuse of knowledge, company needs to pick a client-server model, which will likely be following the "headquarters commissioned and executed" strategy.

If knowledge management strategy goal is to connect the knowledge sources, then peer-to-peer model would be selected. In this case knowledge management system will be decentralized and managed by employees.

If the company decides to use a combined approach, the hybrid model will be used which means having both the central repository for knowledge sharing and encouraging employees to share their knowledge with peers.

In multi-company projects, the generic approach is to use hybrid model, as both companies have their own knowledge management system and it is inefficient to setup a separate knowledge management system for an individual project. In some cases the companies might decide to mandate the use one of their centralized knowledge management system for a shared project. The practically pure client-server model in this case is enforcing the knowledge management strategy of the company that hosts the system.

4.2. Resource allocation issues

Resource allocation is one of the main tasks of the project manager. It begins with initial project planning, when manager has to project which kind of experts will be needed to undertake the project and how many man hours could the project take. The initial rough estimations are later refined as the project moves closer to implementation.

Resource management is never perfect and may need to be adjusted as the project is implemented. Communicating those changes to the customer is a difficult task since in a lot of cases customer has a set budget in mind for the software project.

In multi-company projects the resourcing is way more complicated than in the projects that are managed by a single company. The issues here mostly come form coordination difficulties. They take various forms and require different solutions.

In IEEE paper on resourcing in global projects [Battin et al., 2001] some of these resourcing problems have been analyzed.

The main issues mentioned by the paper are:

- loss of communication richness,
- coordination breakdowns,
- geographic dispersion,
- loss of “teamness”
- and cultural differences.

The aforementioned paper also proposed some solutions to these issues. The communication richness can be maintained with constant exchange of ideas and by putting an effort in keeping the communication alive. This was accomplished by providing representatives for each development center and each domain of expertise. It is also important to keep teams in touch by using conference calls and by allowing people to travel in cases where physical presence was especially important.

Coordination breakdowns can be tackled with the proper workload distribution and by maintaining a clear, simplified version of the software architecture that all developers can refer to. The paper also recommends grouping the work into meaningful areas and integrating parts of the system developed in different places in a careful step-by-step manner.

Software configuration management also was one of the coordination issues mentioned in the article. It was addressed using a software tool that can replicate the configuration data across different sites. This allowed the teams to work independently while seeing the configuration changes made by other teams.

Geographic dispersion was resolved with involvement of vendors, having centralized bug reporting system and careful planning. There were also cases in which special expertise was needed to deal with requests from local authorities.

The loss of “teamness” was described in the paper as a problem when different teams have different development processes. The solution that was chosen for this issue was to give the teams the freedom over which development process they use, but make sure they deliver the result in some standardized form. It was also decided that each team will be required to deliver a complete tested subsystem of a software product, thus avoiding the need of sharing too much internal structure information between teams.

In multi-company projects the issues mentioned by the paper are further complicated by the fact that different teams answer to different management structures. Communication richness is kept by ensuring that teams communicate with each other, even though they are in different companies. Management should take the lead in introducing team members to each other and making sure they also know the managers from the other companies.

Maintaining clear vision and project architecture is extremely important in multi-company projects. This can be achieved by having unified document storage. All team members need to know where to get information on the project structure and on the overall project schedule.

4.3. Dependency issues

The inter-dependency issues are one of the most important issues in complex software projects. In cases when a single company undertakes the project, the dependency issues are managed by a team lead, who has to determine which parts of the project need to be completed first in order for the rest to be completed on time.

In multi-company projects this mechanic no longer works, because different companies can implement different parts of the system. It becomes a task of a project manager to communicate to other teams what needs to be done first and take their advice into account when selecting tasks to implement.

In the paper "Communication and co-ordination practices in software engineering projects" [McChesney and Gallagher, 2004] co-ordination theory is mentioned as a way to formalize co-operative work in complex work settings.

The paper defines several dependency types:

- shared resources,
- producer/consumer relationship,
- prerequisite constraints,
- accessibility,
- usability,
- fit constraints
- and task/subtask dependencies.

Shared resources is a type of dependency in which a certain physical or virtual resource is shared between several team members and cannot be used by several team members at the same time. The paper proposes several ways to resolve this dependency such as: first come first serve, priority order, managerial decision or bidding.

Producer/consumer relationship is a type of dependency when a continued work result of one team member is a prerequisite for the other team member's continued task. This can be resolved by properly timing the dependent tasks.

Prerequisite constraint is a type of dependency in which a certain procedure needs to be followed to continue with the task. The paper suggests that this issue can be resolved using notification, sequencing or tracking.

Accessibility (the right place) constraint is a type of constraint in which a task needs to have certain physical requirements fulfilled. The article suggests that the way to tackle it is inventory management and task allocation.

Usability (the right thing) is a type of constraint that limits the execution of certain tasks via a strict definition. This can be a detailed feature request, required change or some other task that needs to be done at this particular moment. Those type of

constraints can be avoided or accounted for by investing in standardization and practicing participatory design.

Fit constraints are a type of constraint, which dictates certain procedures that need to be followed to ensure project code stability.

Task/subtask dependencies are a type of constraint, which occurs in complex feature implementations and can be dealt with careful goal selection and task decomposition.

In multi-company projects dependencies are managed through the cooperation between project managers. All of the dependency types are further complicated by the fact that they are spanning across several companies.

The use of shared resources needs to be communicated to the project managers of the involved teams. Cross-team producer/consumer relationships need to be properly documented and possibly standardized. Prerequisite constraints need to be properly documented and agreed upon by the project managers from all teams. Accessibility constraints need to be officially documented and agreed upon. Fit constraints need to be agreed upon between project managers and team leads. Usability constraints will require a common meeting if they concern all teams, or at least need to be documented and communicated to other teams if their involvement is not directly required.

4.4. Management cooperation issues

Building management relations is a challenging task in a world of growing economy and globalization. There are numerous ways in which relations between managers or between companies can affect a multi-company project. While there can be nuances in the management relations, one can say that most of cases can be split into two categories: management relations that are harmful to the project and relations that are helpful for the project.

The paper “How should companies interact in business networks?” [Håkansson and Ford, 2001] notes that a complex business market can be seen as a network with businesses as it’s nodes and the relationships between them are threads connecting those nodes. The terms “relationships” and “nodes” are becoming increasingly popular not only in the academic world, but also in the business world itself.

Håkansson and Ford stress that companies should be cautions not only about their relationships with the customer, but also about their relationships with the other companies in the field. Companies should seek a way to inflience their relationships with the others by carefully analysing the costs that the change will involve and the opportunities that it will bring. Companies need to be mindful of the fact that they are not the only ones that can influence the relationship network, this could also be done by their allies or rivals. Even when the company has positioned itself properly in the

business network it needs to carefully analyse what others are doing and encorage them to clarify their goals and their understanding of relationship network.

On the managerial level, relationships with the other companies have to be considered as well. One example would be a project in which a customer needs two really different sets of experts and in this case two companies with good relations that have the proper skills will have an advantage over two rival companies with same skill sets. This basically means that if you are aiming to get a project that requires a skill set your company does posess and the one that it does not, having an allied software house with that skill set will significantly increase your chances of getting that project. With this in mind, project managers should always try to improve their relationships with their counterparts from other software houses, especially if their companies posess different sets of expertise.

In cases where the relationships between companies are bad or difficult, the project manager should try to document the interation between the companies to ensure that the project will not be used to harm the company's reputation. This is especially true if the relationships are bad because of competition between companies.

On the "personal" level, relations between companies can range from non existent to collaboration or fierce competition. In their paper "Cooperation and competition in relationships between competitors in business networks" Bengtsson and Kock [1999] suggest that a company can be in four different types of horizontal relationships at the same time. A company can be competitive, cooperative, symbiotic or both competitive and cooperative.

Bengtsson and Kock [1999] define the following relationship types:

- coexistence,
- cooperation,
- competition
- and co-opetition.

Coexistence consists of solely information and social interactions, it does not involve any financial interaction. Trust is regarded as high, but informal. Interaction norms are strict and informal.

Cooperation is a regular business partnership, which may include informational, social and financial interaction. Trust and norms may be formalized using official agreements.

Competition is a negative action-reaction type of relationship. There is no information sharing and norms are regulated by informal rules-of-play, which are widely accepted.

Co-opetition is a type of relationship that combines cooperation and competition. Two business entities are cooperating in one area and competing in another. Cooperation is based on a formal agreement or on trust, competition is regulated by invisible norms that two businesses have agreed upon.

In multi-company projects, the participating entities will likely be in a co-opetition type of relationship, unless their area of expertise is vastly different.

Hofstede's [2010] cultural dimensions also have significant impact on the company relations. Different cultural backgrounds can cause misunderstandings and conflicts between both team members and managers of the companies. Hofstede provides an example of building relationships between a Saudi engineering firm and Swedish hi-tech corporation, which illustrates how collectivistic and individualistic cultures can matter in contractual relationships between companies.

In terms of society-based culture dimensions, the most important ones here are power distance, collectivistic vs. individualistic and uncertainty avoidance dimensions. Different power distance can cause issues when communicating team members have different positions in their respective company hierarchy, i.e. when a junior developer contacts a senior developer from another company.

Collectivistic cultures tend to regard people outside of their established group with more hostility. It will require some effort from project managers of participating companies to build bridges between teams in this situation.

Companies in which employees come from high uncertainty avoidance cultures will likely demand formalized procedures and standardized documentation, which will cause delays since it will require some negotiations with all involved parties.

Nakata [2009] states that cultures should not be regarded as "rigid" and unchanging, which means that frequent involvement in multi-company projects should bend both corporate and societal cultures adapting them to the diverse environment.

Agile projects have special needs in multi-site and multi-company settings. Agile software development is highly dependent on face-to-face communication. In his thesis Palokangas [2013] stresses that high dependence on direct communication is one of the pitfalls of the agile projects in multi-site development. It is necessary to provide some means to enable face-to-face communication in a distributed team.

4.5. Customer relations issues

Customer relationships are one of the key things in software development. Customers are main stakeholders in the project and they decide on the main features of the resulting software solution. Customer Relations Management (CRM) is an established way of managing customer interactions, it involves gathering and analyzing information about customers in order to sell them more goods or do that more efficiently.

Building customer relations is a challenging task and it becomes especially hard in multi-company projects. Essentially each company has it's own view on the customer relations and it's own history of those relations.

The recent technology developments are making it possible to interact with the customers in a way that was not possible before. In their paper "Understanding customer relationship management" Chen and Popovich [2003] show that CRM has evolved significantly in the last decade. The article names advancements in IT as the main source of those improvements. CRM has evolved from sales force automation (SFA) to become a customer-centric technology which is focused on keeping existing customers rather than getting new customers. In software development, where the market is already saturated with great offers, it is exceedingly important to keep your customers from going away.

In his paper "Customer relationship management: key components for IT success" Bose [2002] suggests that companies will be forced to increasingly switch from managing a market to managing a specific customer.

In multi-company projects, one customer is shared with several software development companies. In this regard, each company needs to think not only about their own relationships with the customer, but also about how customer perceives all of those companies as a group. Ideally, software houses participating in a multi-company project should coordinate their efforts and build the customer relationships together, but it is not always possible, especially in cases where the companies are direct competitors.

5. Case study

In this chapter some of the real-world examples of multi-company projects are described and analyzed. Those cases were observed and documented from a developer point of perspective, so the project manager view on them might differ. It is also important to note that it is hard to get an honest and documented opinion in cases where a project is performed by multiple business entities, some of which are in direct competition with each other. Also, the following case descriptions were deliberately stripped of information that might identify the participating business entities. The information presented here is based on personal experiences, emails and documentation gathered during the implementation of the projects.

The first example case of multi-company cooperation is a project undertaken by two companies in parallel. Company A and B have been working together form the start to plan and execute project tasks. Overall plan was made during the initial meetings and it was followed on both sides with some level of synchronization. The cooperation was initially organized via mailing lists, but later followed up by a Skype chat. Overall, project progressed smoothly, with minor hurdles.

Customer had a moderate influence in relationships between the two companies as many features required synchronized work attempts. The reception of the project was good, customer appreciated contact efforts by both companies and followed project closely. This improved the communication not only between the customer and the software companies, but also helped the software companies cooperate more efficiently between each other.

The following cooperation issues were observed:

- task synchronization issues,
- knowledge sharing issues,
- communication difficulties
- and technical issues

The work effort was mostly synchronized during the initial planning meetings. A set of user story level tasks was approved by the customer and placed in the order of their dependence on each other. Project followed a variant of scrum as a development methodology. Sprints were organized and hosted almost entirely following the Scrum default procedures.

The cooperation issues faced in the project did not cause that much disturbance. Synchronization issues mostly came from the fact that holiday plans were disturbing the work schedule of the other company and there was no proper way found to resolve that. Project managers from both companies tried to inform of the personnel availability changes beforehand, but that did not resolve the issue completely.

Knowledge sharing issues were caused by having several places for documentation storage and by the fact that knowledge area experts were not always reachable. The knowledge transfer was mostly performed by direct communication between developers. In cases where there was no clear information on who to ask about certain problem domain a project manager from the other company's team was inquired about it. Usually the project manager from the other company's team was able to provide contact details for the developer with the required knowledge.

Communication difficulties were caused by the fact that there were two separate teams working on the project and those teams have not met physically. This situation was quite similar to the difficulties each company faced while working on projects distributed geographically between their own offices. Experience in organizing multi-site projects gave project managers from both companies an insight on solving the communication issues in this multi-company project. Work coordination was done via digital communication mechanisms and was in some occasions slow, but it was reliable enough.

Technical issues were caused by the fact that support software and production environments were housed by different companies. Using those tools was complicated and information had to be duplicated in some cases, which created two different sets of knowledge for each company. Those sets of knowledge (mostly issues in the tracking systems) had to be synchronized and maintained manually. Software developers from company A mostly had to copy the necessary information from their systems to the company B's systems. Company B did not have an interest in transferring the information to the company A's systems because the company A's work was independent to some extent.

The issues were resolved with mostly improving direct communication between the companies. Customer was involved during demos and in making overall decisions about project. Task synchronization issues were resolved by a coordinated effort from project managers of both companies. Knowledge sharing issues were mostly resolved on a per-developer level. Communication difficulties were resolved mostly without significant managerial help, but in some cases it was easier to ask the team manager in the other company about work schedule details of a given specialist. Technical issues were largely resolved by direct developer communication, though some of those issues remained unsolved.

There was no reason for integrating the issue management systems, as the companies do not do common projects frequently. With this in mind, no long-term user accounts for external access or shared discussion systems were deemed necessary.

The second example of the multi-company cooperation is a code update project that was undertaken by company A based on the code made by company B. Due to the remaining contractual obligations, company B had to provide support and materials for the project during the initial steps of the implementation. The overall planning was mostly done by company A as the company B's role was solely to provide assistance in understanding the project structure. The project started with a slow pace; initially developers were concentrating on receiving the code from the company B and figuring out how it worked.

The following issues were observed:

- material transfer issues,
- knowledge transfer issues,
- technical issues.

The overall project plan had to be corrected in the first months of project implementation, mostly because material transfer was delayed and there were technical issues with access to third party servers.

After the initial difficulties, the project was still often plagued by knowledge transfer issues. The main reason for this was that not much of the original team that implemented this project remained in company B.

Technical issues were also causing all sorts of problems since some of them had to be communicated to company B specialists via email. The answers were not always on time and some of the questions were not answered fully because of the lack of expertise on company B’s side.

Reporting and communication surprisingly were not much of an issue in this case, since there was minimal need of managerial involvement and emails were answered promptly. Most of the issues requiring managerial involvement were related to communication with customer or hosting provider support. Issue management was done primarily in company A’s own system, this knowledge was not shared with company B’s personnel because there was no need for it.

The issues in this case were resolved mostly by careful planning and making sure that all issues that require company B's support were resolved before the support contract with the company B ended.

In the third case, the project involved integrating a software component developed by company B into an existing software solution made by company A. The project proved to be a quite challenging endeavor because of almost non-existent cooperation from company B in regards to their component. The component itself was a quite complex piece of software, which included 3 servers and required extensive configuration work.

Following issues were observed in this case:

- knowledge transfer issues,
- technical issues,
- and communication issues.

The project was quite challenging throughout the whole implementation. The materials received from company B included source code, built binaries and a set of documentation. Provided material looked really convincing and it was assumed that it should not be a hard task to make required integrations. However, the reality proved otherwise, since the materials were riddled with small issues. One of the examples would be that installation instructions were mentioning libraries that were supposed to be placed on a server, but neither the exact library list nor the folder were those needed to be was not specified.

The issues in this case were resolved mostly via direct managerial contact with the customer. The company B refused to provide support, citing that it would require a support contract to be signed. Eventually, most of the critical questions were asked by the customer on behalf of the company A. The technical issues were resolved slowly by company A's specialists, because the company B personnel answered only the absolutely critical questions and did that quite slowly.

6. Discussion

6.1. Summary

The real world cases that were described in the case study chapter correlate well with the reviewed theoretical studies. In this chapter we will discuss the applicability of the solutions mentioned in the literature review and their possible applications to the real-world projects.

Reporting issues in multi-company projects have proven to be manageable, given that all participating software houses are willing to investigate and find solution for those issues. Report and knowledge sharing is an important part of the project and the participating parties have to agree beforehand on solutions that they are going to use in that area. Literature review suggests that individual companies can choose different approaches at knowledge management and sharing, but in case of the multi-company project choices become quite limited. Depending on the type of the participation, there can be different solutions to the knowledge sharing problem, but in cases where software houses stand on equal grounds the client-server model works the best. All the cases reviewed in the previous chapter had some kind of client-server knowledge sharing system in place. In cases where the information needed to be shared with the other participating software house an agreement was made regarding the common knowledge sharing platform. In one of the cases issues had to be stored in two issue-tracking systems, but that was only the case for the handful of issues that were relevant for both companies. Knowledge sharing issues observed in the case study chapter were of two kinds: knowledge sharing strategy issues and lack of expertise. The lack of expertise could not be resolved because the experts were either not available or demanded extra contractual obligations (in the 3rd case). The knowledge sharing strategy issues, however, could have been resolved if the companies would be willing to put more effort towards creating a shared knowledge database.

Resource allocation issues have proven to be a big problem in the multi-company projects, mostly because it is hard to allocate a resource, which belongs to another business entity. From the issues mentioned in the literature review, the ones that appeared to be most noticeable in the case studies were: loss of communication richness, coordination breakdowns and the loss of “teamness”. The loss of the communication richness is a big issue in multi-company projects, since most of the conversations between different teams has to be done via teleconference software. In some cases developers never see their counterparts or hear their voice. The coordination breakdown issues have not been noticed that much in the case studies, most of them

were actually caused by small initial planning mistakes. All of the coordination issues that did appear were solved quite swiftly, except for case 3 where the other company refused to cooperate most of the time. Loss of the "teamness" is almost a usual characteristic of a multi-company project, since different companies have different development practices. There was almost no attempt to tackle this issue in the cases reviewed in Chapter 5, since there was no practical need for it. Loss of "teamness" would be an issue in a multi-company project only if companies have to synchronize their development practices really tightly for some reason. As and example: if they are contractually required to have teams, which include developers from all companies. Geographic dispersion was not a significant issue in the case study samples, but it would be otherwise a big deal in an international multi-company project.

Management cooperation is an important matter in the multi-company project environment. Building good relationships between companies is an obligation of a project manager if he/she is involved in a multi-company project. From the cases in Chapter 5: first case is a good example of positive company relationships (co-opetition); in second case company relationships were almost non-existent because of project circumstances (coexistence); in the third case the relationships were initially hostile, but improved slightly afterwards. The handling of the inter-company relationships was quite simplistic and mostly had to do with trying to improve the co-operation between companies. The deep strategic goals of building a business network (as mentioned in the literature review chapter) were not the primary concern of the project managers in case study examples. It seems that during the implementation of the project managers are mostly concerned with immediate customer and cross-company relationships at hand, rather than trying to improve the company's standing in the business network.

Improving customer relationships is a cornerstone of the project management efforts. One can say that one of the goals of the software project is making sure customer will order more next time. In the case studies customer relations were on quite a good level. Participating companies shared a belief that direct customer participation in the project improves the quality of the project significantly and ensures smooth progress. This conviction was mutual as customer asked to be involved in testing and even in some developer discussions. In the literature review it was revealed that more and more companies decide to adopt CRM and try to manage each customer individually. This certainly is true for smaller software development companies, which feel the need to keep their clients due to the maturity of the market and high levels of competition.

6.2. Research limitations and future directions

In this thesis the goal was to analyze the interactions between two or more software houses participating in a project on equal standing. This does not cover other possible scenarios, like having a great difference in company sizes between the project participants or working on an open source project and interacting with the community.

Chosen research method (literature review) is a limiting factor since many of the aspects of multi-company interaction may not be properly extrapolated from the multi-site software development papers. This research area also does not provide too much room for openness, as such it will be beneficial to have a non-profit organization or an independent group of experts analyze several such cases with proper documentation and a decent number of sample cases. Future research on this subject may also benefit from having properly organized and held systematic literature review on related topics mentioned in this thesis, such as:

- business relations,
- customer relations,
- global software development issues,
- and human resource management in global setting.

While it is possible to look at the multi-company project interaction from the inside, it is really hard to make a case out of that experience because you are not able to disclose much of the details.

It would also be beneficial to analyze different types of inter-company relationships in the multi-company project. Equal partnership is only one of the ways companies can interact within a project, with other interesting cases including subcontracting relationships, support service contract relationships et cetera.

Other areas that were not specifically targeted by this thesis include: software quality assurance, cross-border relationships in multi-company projects, dealing with unexpected situations et cetera.

Additionally, it would be beneficial to investigate customer's issues when preparing and following a multi-company software project. There are many co-operation issues that customer has to take care of, some of which are done on behalf of one of the participating software houses.

Multi-company projects are an interesting research topic and there is still much to learn about different interactions between companies and people that can occur in them. My personal goals for conducting this research was to analyze the issues I have personally encountered while working in several multi-company project and try to formulate solutions to the common problems encountered. The goal of this thesis was to investigate the relationships between companies in the situation when they have to work

together to reach a shared objective. There was not much direct research available on multi-company software development, so multi-site development was chosen for literature review in an attempt to extrapolate issues found in multi-site development to multi-company development. Some research articles were taken from related fields of project management, business management and industrial management.

After reviewing the literature on the multi-site projects, management cooperation and customer relation management the situations in multi-company projects became easier to predict. Some of the problems of multi-site projects had been given a new dimension in multi-company projects and not all solutions proposed in multi-site projects research papers worked in multi-company project environment.

A unique quality of multi-company projects is that they have several independent managing hierarchies. This is the biggest reason why a solution for multi-site development problem might not work in a multi-company project. The only way to overcome this difficulty is careful planning and good inter-company relationships.

Good level of communication between project managers and team members across several companies is also a way to tackle the multi-company project issues. Project managers have to organize frequent meetings and make sure that team members know the knowledge area experts in the other companies and how to reach them.

Knowledge sharing has to be planned in the very beginning of the project. A good strategy is to agree on using an existing knowledge sharing system of one of the participating companies. Issue tracking should preferably be unified as well, unless each team's tasks have minimum dependency on each other.

Even if the tasks have minimum coupling and can be done separately, it is extremely important to keep partners informed of the progress of your team. It is not only good for tracking the overall completion of the project, but also to eliminate accusations of stalling the project.

To conclude, direct and honest communication seems to be the main solution for multi-company project issues.

References

[Alfresco, 2014] Alfresco, Alfresco document management software, website, 2014. *http://www.alfresco.com*

[Atlassian, 2014] Atlassian, Atlassian software: Jira and Confluence, website, 2014. *https://www.atlassian.com/software*

[Baldwin, 2014] Howard Baldwin, Time off to innovate: Good idea or a waste of tech talent?, website, 2014. *http://www.computerworld.com/s/article/9229373/Time_off_to_innovate_Good_idea_or_a_waste_of_tech_talent_*

[Battin et al., 2001] Robert D. Battin, Ron Crocker, Joe Kreidler and K. Subramanian, Leveraging Resources in Global Software Development, *IEEE Software*, 2001, 70-77.

[Bengtsson and Kock, 1999] Maria Bengtsson and Sören Kock, Cooperation and competition in relationships between competitors in business networks, *Journal of Business and Industrial Marketing* **14** (3), 1999, 178-193.

[Benington, 1982] Herbert D. Benington, Production of Large Computer Programs, *IEEE Annals of the History of Computing* **5** (4), 1983, 350–361.

[Bose, 2002] Ranjit Bose, Customer relationship management: key components for IT success, *Industrial Management & Data Systems* **102** (2), 2002, 89-97.

[Chemuturi and Cagley, 2010] Murali K. Chemuturi and Thomas M. Cagley Jr., *Mastering Software Project Management: Best Practices, Tools and Techniques, 2010.*

[Chen and Popovich, 2003] Injazz J. Chen and Karen Popovich, Understanding customer relationship management, *Business Process Management Journal* **9** (5), 2003, 672-688.

[Computerworld, 2001] Computerworld, Extreme Programming, website, 2001. *http://www.computerworld.com/s/article/66192/Extreme_Programming*

[Desouza et al., 2006] Kevin C. Desouza, Yukika Awazu and Peter Baloh, Managing Knowledge in Global Software Development Efforts: Issues and Practices, *IEEE Software **23** (5)*, 30-37, September/October 2006, 30-37.

[Dingsøyr and Šmite, 2014] Torgeir Dingsøyr and Darja Šmite, Managing Knowledge in Global Software Development Projects, *IEEE IT-Pro*, January/February 2014, 22-29.

[Martin, 1991] James Martin, *Rapid Application Development, MacMillan, 1991.*

[Herbsleb and Moitra, 2001] James D. Herbsleb and Deependra Moitra, Global Software Development. *IEEE Science **18** (2),* March/April 2001, 16-20.

[Hofstede, 2010] Geert Hofstede, Gert Jan Hofstede and Michael Minkov, *Cultures and Organizations: Software of the Mind (3rd ed.), McGraw-Hill, 2010.*

[Hughes and Cotterell, 1999] Bob Hughes and Mike Cotterel, *Software Project Management (2nd ed.), Cambrige, 1999.*

[Håkansson and Ford, 2001] Håkan Håkansson and David Ford, How should companies interact in business networks?, *Journal of Business Research* **55**, 2002, 133-139.

[McChesney and Gallagher, 2004] Ian R. McChesney and Séamus Gallagher, Communication and co-ordination practices in software engineering projects. *Information and Software Technology* **46**, 2004, 473-489.

[Mockus and Weiss, 2001] Audris Mockus and David M. Weiss, Globalization by Chunking: A Quantitative Approach, *IEEE Software*, March/April 2001, 30-37.

[Nakata, 2009] Cheryl Nakata, *Beyond Hofstede: Culture Frameworks for Global Marketing and Management, Palgrave Macmillan, 2009.*

[Nguyen-Duc et al., 2014] Anh Nguyen-Duc, Daniela S. Cruzes and Reidar Conradi, The impact of global dispersion on coordination, team performance and software quality – A systematic literature review, *Information and Software Technology*, Available online 18 June 2014, ISSN 0950-5849.
http://dx.doi.org/10.1016/j.infsof.2014.06.002

[Palmer and Felsing, 1999] Stephen R. Palmer and John M. Felsing, *A Practical Guide to Feature-Driven Development, Prentice Hall., 2002.*

[Palokangas, 2013] Jaakko Palokangas, Agile Around the World – How Agile Values Are Interpreted in National Cultures?, *MSc Thesis - University of Tampere*, 2013.

[Phan et al., 1995] Dien D. Phan, Joey F. George and Douglas R. Vogel, Managing software quality in a very large development project, *Information & Management* **29**, 1995, 277-283.

[Pikkarainen et al., 2008] Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson and Jari Still, The impact of agile practices on communication in software development, *Information and Software Technology* **52**, 2010, 779-791.

[Prikladnicki and Audy, 2010] Rafael Prikladnicki and Jorge Luis Nicolas Audy, Process models in the practice of distributed software development: A systematic review of the literature, *Information and Software Technology* **52**, 2010, 779-791.

[PRINCE2, 2001] PRINCE2 Official Website, website, 2014.
http://www.prince-officialsite.com

[Project Management Institute, 2013] Project Management Institute, *A guide to Project Management Body Of Knowledge (5th ed.)*, Project Management Institute, Inc., 2013.

[Scrum Institute, 2014] Scrum Institute, Scrum effort estimations – planning poker, website, 2014. *http://www.scrum-institute.org/Effort_Estimations_Planning_Poker.php*

[Thayer et al., 1980] Richard H. Thayer, Arthur B. Pyster and Roger C. Wood, Major Issues in Software Engineering Project Management, IEEE Transactions on Software Engineering **7** (4), July 1981, 51-59.

[Verner et al., 2014] J.M. Verner, O.P. Brereton, B.A. Kitchenham, M. Turner and M. Niazi, Risks and risk mitigation in global software development: A tertiary study, Information and Software Technology 56 (1), January 2014, 54-78.